Database Migrations - LiquidBase Workflow

Project: Signature Router & Management System
Database: PostgreSQL 15
Migration Tool: LiquidBase 4.x
Last Updated: 2025-11-26

📋 Overview

Este proyecto usa **LiquidBase** para gestionar migraciones de base de datos siguiendo estándares corporativos. LiquidBase garantiza:

- ✅ **Versionado de schema** (cada cambio rastreado en `DATABASECHANGELOG`)
- ✅ **Idempotencia** (changesets no se ejecutan dos veces)
- ✅ **Rollback declarativo** (cada changeset incluye rollback)
- ✅ **Multi-entorno** (contexts: dev/uat/prod)

🗂️ Directory Structure

```
src/main/resources/liquibase/
├── changelog-master.yaml           # Master changelog (includeAll per
environment)
└── changes/
    ├── dev/                         # DEV changesets (context: dev)
    │   ├── 0001-create-uuidv7-function.yaml
    │   ├── 0002-create-signature-request-table.yaml
    │   ├── 0003-create-signature-challenge-table.yaml
    │   ├── 0004-create-routing-rule-table.yaml
    │   ├── 0005-create-connector-config-table.yaml
    │   ├── 0006-create-outbox-event-table.yaml
    │   └── 0007-create-audit-log-table.yaml
    ├── uat/                         # UAT changesets (context: uat)
    │   └── (same files as dev/)
    └── prod/                        # PROD changesets (context: prod)
        └── (same files as dev/)
```

📝 ChangeSet Standards (Corporate)

Mandatory Fields

Cada changeset **DEBE** incluir:

1. `id`: Numérico consecutivo (0001, 0002, ...) para ordenamiento alfabético
2. `author`: Nombre + email del autor
3. `context`: Entorno específico (`dev`, `uat`, o `prod`)
4. `changes`: Lista de cambios (createTable, addColumn, etc.)
5. `rollback`: **MANDATORY** (especialmente crítico en prod)

YAML Format

Preferir YAML sobre XML/SQL por legibilidad y rollback declarativo.

Ejemplo:

```yaml
databaseChangeLog:
  - changeSet:
      id: "0002"
      author: "BMAD Dev Agent <bmad@signature-router.com>"
      context: dev
      comment: "Create signature_request table - Aggregate root"
      changes:
        - createTable:
            tableName: signature_request
            columns:
              - column:
                  name: id
                  type: uuid
                  defaultValueComputed: uuid_generate_v7()
                  constraints:
                    primaryKey: true
                    nullable: false
              # ... more columns
        - createIndex:
            indexName: idx_signature_request_customer_id
            tableName: signature_request
            columns:
              - column:
                  name: customer_id
      rollback:
        - dropTable:
            tableName: signature_request
```

```
            cascade: true
```

GIN Indexes (JSONB)

LiquidBase YAML `createIndex` **NO soporta** `USING GIN`. Usar SQL raw:

```
changes:
  - sql:
      sql: "CREATE INDEX idx_signature_request_context_gin ON signature_request
USING GIN (transaction_context);"
```

🔄 Promotion Flow (DEV → UAT → PROD)

Step 1: Develop in DEV

1. Crear changeset en `changes/dev/NNNN-descriptive-name.yaml` con `context: dev`
2. Incluir **rollback block** (MANDATORY)
3. Test local: `./mvnw spring-boot:run -Dspring.profiles.active=local`
4. Verificar con `psql`: `\dt` (listar tablas), `\d table_name` (describe tabla)

Step 2: Promote to UAT

1. Copiar changeset a `changes/uat/`
2. **Cambiar** `context: dev` **a** `context: uat`
3. Commit en branch `uat`
4. Pipeline CI/CD ejecuta changeset en BD UAT
5. Validar con integration tests en entorno UAT

Step 3: Promote to PROD

1. Copiar changeset a `changes/prod/`
2. **Cambiar** `context: uat` **a** `context: prod`
3. Commit en branch `main/release`
4. **Manual approval** required en pipeline CI/CD
5. Pipeline ejecuta changeset en BD PROD
6. Validar con smoke tests

CRITICAL: Rollback debe estar probado en UAT antes de PROD.

🚀 Running Migrations

Local Development (Docker Compose)

```
# 1. Start PostgreSQL
docker-compose up -d postgres

# 2. Verify PostgreSQL is ready
docker-compose logs -f postgres
# Wait for: "database system is ready to accept connections"

# 3. Run application (LiquidBase auto-executes)
./mvnw spring-boot:run -Dspring.profiles.active=local

# 4. Verify changesets executed
docker exec -it signature-router-postgres psql -U siguser -d signature_router \
  -c "SELECT id, author, filename FROM databasechangelog ORDER BY orderexecuted;"
```

Manual LiquidBase Commands

```
# Check LiquidBase status
./mvnw liquibase:status

# Rollback last N changesets
./mvnw liquibase:rollback -Dliquibase.rollbackCount=1

# Generate SQL without executing (dry-run)
./mvnw liquibase:updateSQL

# Clear checksums (if YAML was edited after execution)
./mvnw liquibase:clearCheckSums
```

🧪 Testing Strategy

Integration Tests (Testcontainers)

Test class: `DatabaseSchemaIntegrationTest.java`

```
@SpringBootTest
@Testcontainers
class DatabaseSchemaIntegrationTest {
    @Container
    static PostgreSQLContainer<?> postgres = new PostgreSQLContainer<>
("postgres:15-alpine");

    @Test
```

```
    void testAllTablesExist() { /* ... */ }

    @Test
    void testUuidV7FunctionExists() { /* ... */ }

    @Test
    void testJsonbColumnWorks() { /* ... */ }
}
```

Run tests:

```
./mvnw verify
```

Tests verifican:

- ✅ 6 tablas de negocio + 2 LiquidBase tables existen
- ✅ UUIDv7 function retorna UUID sortable
- ✅ JSONB columns soportan queries con `->>` operator
- ✅ FK constraints con CASCADE delete funcionan
- ✅ CHECK constraints se aplican
- ✅ 7 changesets ejecutados en orden correcto

Manual Rollback Test

```
# 1. Verify current state
docker exec -it signature-router-postgres psql -U siguser -d signature_router -c
"\dt"

# 2. Rollback last changeset
./mvnw liquibase:rollback -Dliquibase.rollbackCount=1

# 3. Verify table deleted (e.g., audit_log)
docker exec -it signature-router-postgres psql -U siguser -d signature_router -c
"\dt"

# 4. Re-run application (recreates table)
./mvnw spring-boot:run -Dspring.profiles.active=local

# 5. Verify table recreated
docker exec -it signature-router-postgres psql -U siguser -d signature_router -c
"\dt"
```

📊 LiquidBase Tables

DATABASECHANGELOG

Rastrea changesets ejecutados:

```
SELECT id, author, filename, orderexecuted, exectype
FROM databasechangelog
ORDER BY orderexecuted;
```

Columns:

- `id`: Changeset ID (e.g., "0001", "0002")
- `author`: Autor del changeset
- `filename`: Path al archivo YAML
- `orderexecuted`: Orden de ejecución (1, 2, 3, ...)
- `exectype`: EXECUTED | RERAN | SKIPPED
- `md5sum`: Checksum del changeset (detecta cambios)

DATABASECHANGELOGLOCK

Gestiona locks durante ejecución:

```
SELECT * FROM databasechangeloglock;
```

Columns:

- `locked`: true | false
- `lockgranted`: Timestamp cuando se adquirió lock
- `lockedby`: Host que adquirió lock

Si lock queda stuck:

```
./mvnw liquibase:releaseLocks
```

⚠️ Common Issues & Solutions

Issue 1: ChangeSet Already Exists

Error:

```
Changeset changes/dev/0002-create-signature-request-table.yaml::0002::author has already been run
```

Cause: LiquidBase detecta changeset ya ejecutado (checksum en `DATABASECHANGELOG`).

Solution:

- Esto es comportamiento **esperado** (idempotencia).
- LiquidBase **NO** ejecuta changesets dos veces.
- Si necesitas modificar tabla, crea **nuevo changeset** (e.g., `0008-add-column-to-signature-request.yaml`).

Issue 2: Checksum Validation Failed

Error:

```
Validation Failed: 1 changeset(s) check sum
```

Cause: Archivo YAML fue editado después de ejecutarse.

Solution:

```
# Clear checksums (ONLY in dev)
./mvnw liquibase:clearCheckSums

# Re-run application
./mvnw spring-boot:run
```

⚠️ **NEVER clear checksums en PROD.**

Issue 3: Rollback Failed

Error:

```
No inverse action found for createTable
```

Cause: Changeset no tiene bloque `rollback:`.

Solution:

- **Siempre incluir rollback block** (corporate standard).
- LiquidBase puede inferir rollback para algunos cambios (e.g., `createTable` → `dropTable`), pero **explícito es mejor**.

Issue 4: PostgreSQL Not Ready

Error:

```
org.postgresql.util.PSQLException: Connection refused
```

Cause: App inició antes de que PostgreSQL esté ready.

Solution:

- Docker Compose healthcheck debe estar configurado:

```yaml
healthcheck:
  test: ["CMD-SHELL", "pg_isready -U siguser"]
  interval: 10s
  timeout: 5s
  retries: 5
```

- Esperar logs: "database system is ready to accept connections"

🔒 Security & Compliance

PII Protection (GDPR)

- `customer_id`: Pseudonimizado (NO PII directo)
- `transaction_context`: JSONB puede contener business data, **NO PII plain text**
- **Encryption-at-rest**: TDE (Transparent Data Encryption) habilitado en prod

Non-Repudiation (PCI-DSS)

- `signature_challenge.provider_proof`: Almacena cryptographic receipt de provider (e.g., Twilio Message SID)
- `audit_log`: Rastrea todos los cambios con `user_id`, `ip_address`, `changes` JSONB

Audit Trail (SOC 2)

- Tabla `audit_log` captura:
  - **Who**: `user_id`
  - **What**: `entity_type`, `entity_id`, `action`
  - **When**: `created_at`
  - **Details**: `changes` JSONB con before/after values

📚 References

- LiquidBase Documentation
- PostgreSQL 15 Documentation
- Testcontainers PostgreSQL
- **Internal Docs:**
  - `docs/architecture/03-database-schema.md` - Complete schema DDL
  - `docs/sprint-artifacts/tech-spec-epic-1.md` - LiquidBase migration strategy

Status: ✅ Story 1.2 Complete
Last Migration: 0007-create-audit-log-table.yaml
Next Story: 1.3 (Kafka Infrastructure & Schema Registry)